VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Návrh repositáre studijních projektu se zamerením na rozpoznávání podobných prací

Student's projects repository oriented to searching similar projects

2009 Pavel Týn

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2009 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Rád bych poděkoval Ing. Radoslavovi Fasugovi za odborné vedení, pomoc a čas který mi věnoval.

Abstrakt

Cílem bakalářské práce je vytvořit počítačový program pro automatické porovnávání odevzdaných prací a upozornit na takové práce, které byly pravděpodobně opsány od jiného studenta.

Program umí porovnávat základní údaje souborů jednotlivých prací, jako je jeho název, velikost a datum posledních úprav. Dále umí zvládat porovnat základní parametry dokumentů, mezi které se řadí například autor, titul, apd. Je také schopen porovnat obsah jednotlivých textových souborů a upozornit na podobnost i v případě, že student některé slova, věty či dokonce odstavce odstranil, přesunul nebo nahradil jinými slovy. Součástí aplikace je také práce s archivy.

Výstupem z aplikace tvoří seznam prací podezřelých z toho, že jsou opsány jiným studentem a procento jejich podobnosti.

Aplikace komunikuje s uživatelem pomocí grafického rozhraní.

Klíčová slova: Java, porovnání projektů, XML, POI, Open Document, PDFBox, StringMath

Abstract

The intention of the bachelor´s degree work/study is to create a computer software aiming to compare hand in works and determine such works that have been extracted from other student´s works.

This software can compare basic datas of the particular works like their title, size and date of the recent changes made. Further it allows to compare primary parameters of files, for example name of author, title tec. Also this software can compare contents of particular text files and give an alert notice despite of the fact that some words, sentences or even paragraphs were removed, deleted or replaced by a student. A part of the aplication there is also ability to work with fines. The program output consists of the list of works suspected to be duplicated by the other student, also it will give a percentage of their similarity.

The aplication communicates with user thanks to the graphic interface.

Keywords: Java, project comparing, XML, POI, Open Document, PDFBox, StringMath

Seznam použitých zkratek a symbolů

Java SE	–	Java Platform Standard Edition
JVM	–	Java Virtual Machine - virtuální stroj Javy
API	–	Application Programming Interfaces - rozhraní pro programování aplikací
JRE	–	Java SE Runtime Environment
JDK	–	Java SE Development Kit
javac	–	kompilátor jazyka java
GUI	–	Graphical User Interface - grafické uživatelské rozhraní
AWT	–	Abstract Window Toolkit
OSS	–	Open-source Software
SWT	–	StandardWidget Toolkit
UML	–	Unified Modeling Language
XML	–	EXtensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Nedílnou součástí výuky, především na vysokých školách, je také odevzdávání vlastních vypracovaných projektů a prací, nad kterými si studenti procvičí a osvojí probíranou látku. Aby tyto práce měly smysl, musí být vypracovány samostatně. Při velkém počtu studentů pedagog může jen ztěží kontrolovat, zda-li odevzdaná práce je, či není opsána od jiného studenta. Vyučuje-li stejný předmět více pedagogů, je tato kontrola téměř znemožněna, stejně jako kontrola podobnosti s prácemi z minulých ročníků.

Hlavní motivací mojí bakalářské práce bylo zautomatizovat tyto úkony a používat přitom jednu aplikaci, která poskytne uživateli komfort uživatelského rozhraní. Důraz bude kladen na širokou škálu možností výběru, podle kterých uživatel má možnost upravit nastavení a důležitost jednotlivých analýz a zvýšit tak účinnost celé aplikace.

Vytvořený počítačový program, který jsem nazval JProjectComparator, je tedy určen především pedagogům základních, středních a vysokých škol. JProjectComparator bude umět hromadně vyhodnocovat odevzdané projekty a upozorňovat na projekty, které mohou být podobné a napomoci jim tak k samostatnému řešení úloh.

Práce je rozdělena do 5-ti částí, ve kterých poskytnu čtenáři ucelený pohled na vývoj programu JProjectComparator.

První část se zabývá použitými technologiemi, popisem externích knihoven a jejich využití v mojí práci.

Obsahem druhé části je návrh nástroje. Ten poskytuje pohled na aplikaci potřebný k její následné implementaci.

Třetí část se týká podrobné analýzy systému, která má za úlohu pomoct dostatečně pochopit problematiku daného problému.

Ve čtvrté části se věnuji výše zmíněné implementaci a testování aplikace.

V poslední části najde čtenář návod k instalaci a obsluze programu JProjectComparator.

2 Analýza

V této fázi je důležité správně vymezit hranice problémové oblasti, kterou se zabývám, určit funkční požadavky na systém a oblast popsat. Datovou vrstvu aplikace zachytím vhodným analytickým diagramem.

2.1 Stručný popis systému

Systém se bude skládat ze dvou hlavních logických částí - **scanner** a **analyser**. Scanner bude mít na starosti načtení domácích úloh z repositáře, do kterého posílají studenti své projekty. Poté scanner vyextraktuje informace o projektech do externích XML souborů do vybraného adresáře. Analyse pak má za úkol tyto a případné další XML soubory obsažené v tomto adresáři s projekty načíst a porovnat. Výstupem pak bude seznam podezřelých projektů s procentem podobnosti. Blokové schéma funkce systému je znázorněno na Obrázku 1.

2.2 Vymezení hranic a specifikace požadavků na systém

Pro namodelování tohoto problému je vhodným nástrojem diagram použití případů - *Use Case Diagram* [6]. Uvnitř modelovaného systému se nachází jednotlivé případy užití. Mimo systém stojí takzvaní aktéři - *Actors*, kteří interaktují se systémem. Mohou to být role osob, externí systémy nebo jiné komunikující subjekty. Případ užití se dále specifikuje pomocí toků událostí nebo scenářů, přičemž ve specifikaci uvedeme i název případu použití aktéra.

V případě mé aplikace definujeme v jejím diagramu použití jen jednoho aktéra - **Uživatel**. Ten představuje všechny možné uživatele aplikace.

V původní úvaze následoval případ užití **Nastavení analýzy** teprve až za případem užitít **Spuštění scanneru**. Předpokladem je spuštění aplikace na velké množství projektů, a tedy samotné scannování a analýza může trvat několik hodin. Takto zařazený případ užití by tedy znamenal pozastavení aplikace po dokončení scanneru, a spuštění analýzy by muselo počkat na zásah uživatele. V případě, že se takovéto úlohy budou spouštět např. přes noc, mohla by tato mezipauza značně navýšit dobu trvání pro provedení celého programu.

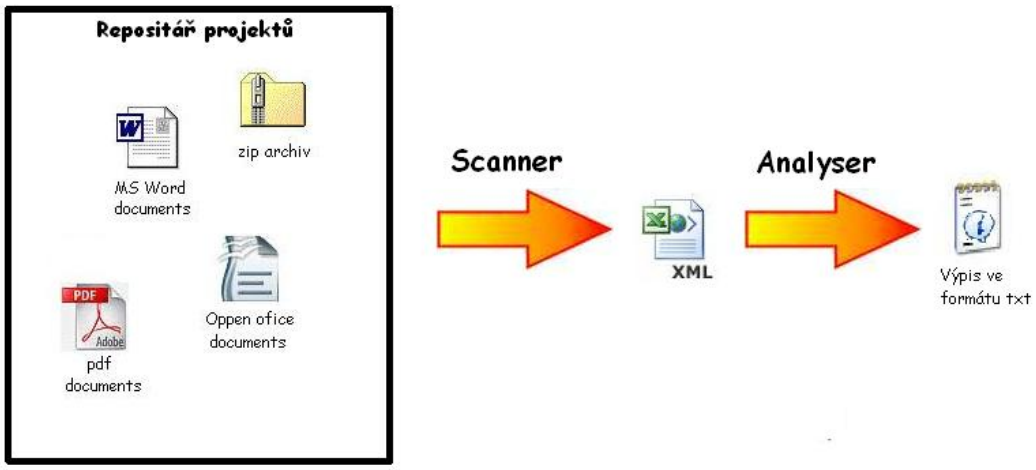
Pro eliminaci tohoto problému jsem se rozhodl zařadit tento případ užití ještě před spuštění scanneru, a sice za případ užití **Nastavení scanneru**.

Případy užití **Spuštění scanneru** a **Spuštění analýzy** lze kdykoliv pozastavit tlačítkem **Pozastavit** a opět spustit pomocí tlačítka **Pokračovat**.

Analýza bude možné spustit nezávisle na scanneru.

2.2.1 Diagram případu užití

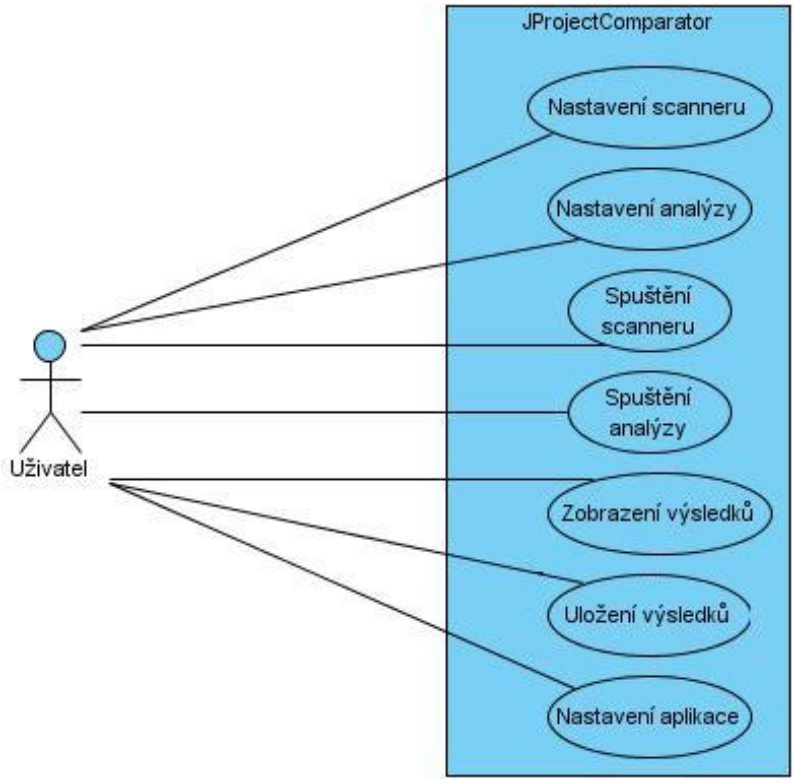
Pro aplikaci bude třeba navrhnout případy užití pro nastavení a provedení scanneru, který ukládá informace z projektů do XML souborů, případy užití pro výběr a provedení



Obrázek 1: Blokové schéma funkce aplikace JProjekctComparator

analýz nad zadanými projekty, dále pak pro zobrazení a uložení výsledku analýzy a pro nastavení celé aplikace.

Diagram případu užití je zobrazen na Obrázku 2.



Obrázek 2: Diagram případu užití pro aplikaci JProjekctComparator

2.2.2 Specifikace případů použití

Ve všech případech užití je jeden stejný aktér - Uživatel a PU znamená případ užití.

Případ užití: Nastavení Scannaru
Vstupní podmínky: Všechny zkoumané projekty jsou uloženy do jednoho **pracovního adresáře** a každý projekt je uložen ve svém adresáři.
Tok událostí: 1. PU začíná při spuštění programu. 2. Aplikace vyžaduje zadat cestu k pracovnímu adresáři, ve kterém jsou uloženy projekty, nebo zaškrtnout políčko **Přeskočit scannování**. 3. Aplikace umožňuje vybrat cestu pro uložení xml souborů, do kterých scanner ukládá informace o projektech. 4. Uživatel zadá cestu k pracovnímu adresáři. 5. Aplikace načte pracovní adresář, zobrazí počet projektů nacházejících se v něm a zpřístupní tlačítko **Pokračovat**. 6. Uživatel dle potřeby zadá cestu k adresáři, do kterých se budou ukládat XML soubory. Defaultně tato hodnota bude nastavena na pomocný adresář aplikace. **Výstupní podmínky**: Uživatel zadal cestu k pracovnímu adresáři.
Alternativní tok: 1. Uživatel zvolil volbu **Přeskočit scannování**. 2. Aplikace zpřístupní tlačítko **Přeskočit** **Výstupní podmínky**: Uživatel zaškrtnul políčko **Přeskočit scannování**.

Případ užití: Nastavení Analýzy
Vstupní podmínky: Uživatel vybral pracovní adresář, nebo zaškrtnul políčko **Přeskočit scannování**.
Tok událostí: 1. PU začíná jakmile Uživatel poklepne na tlačítko **Pokračovat** nebo **Přeskočit** PU Nastavení scanneru. 2. Aplikace umožňuje nastavit cestu k adresáři, ve kterém jsou uloženy XML soubory obsahující informace o projektech. Defaultně je cesta nastavena na pomocný adresář aplikace, případně na adresář, do kterého scanner uložil XML soubory (byl-li spuštěn). 3. Aplikace umožňuje vybrat druhy analýz, které se budou provádět. 4. Uživatel vybere cestu k adres. 5. Aplikace načte pracovní adresář, zobrazí počet projektů nacházejících se v něm a zpřístupní tlačítko **Pokračovat**. 6. Uživatel dle potřeby zadá cestu k adresáři, do kterých se budou ukládat XML soubory. Defaultně tato hodnota bude nastavena na pomocný adresář aplikace. **Výstupní podmínky**: Uživatel nastavil (nebo ponechal přednastavenou) cestu k adresáři s XML soubory a vybral analýzy, které chce provést.

Případ užití: Spuštění scanneru
Vstupní podmínky: Uživatel zadal cestu k pracovnímu adresáři a nevybral možnost **Přeskočit scannování**.
Tok událostí: 1. PU začíná jakmile Uživatel poklepne na tlačítko **Spustit** v PU Nastavení analýzy. 2. Aplikace postupně ukládá informace o každém projektu do samostatného XML souboru. **Výstupní podmínky**: Aplikace zaznamená že scannování bylo dokončeno.
Alternativní tok: 1. Scannování lze kdykoliv přerušit. **Výstupní podmínky**: Aplikace se ukončí.

Případ užití: Spuštění alanýzy
Vstupní podmínky: Uživatel vybral možnost **Přeskočit scannování** nebo bylo úspěšně dokončeno scannování.
Tok událostí: 1. PU začíná, když aplikace zaznamená ukončení scannování. V případě že Uživatel vybral možnost **Přeskočit scannování** v PU Nastavení scanneru, spuští se PU poklepnutím na tlačítko **Spustit** v PU Nastavení analýzy. 2. Aplikace načte informace uložené v externích XML souborech z adresáře jež byl zadán v PU Nastavení analýzy. 3. Aplikace porovná jednotlivé projekty mezi sebou vybranými analýzami a ukládá si procento podobnosti s ostatními projekty. **Výstupní podmínky**: Aplikace zaznamená že analýza byla dokončena.
Alternativní tok: 1. Analýzu lze kdykoliv přerušit. **Výstupní podmínky**: Aplikace se ukončí.

Případ užití: Zobrazení výsledků
Vstupní podmínky: Analýza byla úspěšně dokončena.
Tok událostí: 1. PU začíná, když aplikace zaznamená dokončení analýzy. 2. Aplikace zobrazí takové projekty, které překročili limit procenta podobnosti u jednotlivých analýz a jsou tedy podezřelé z kopírování. 3. Aplikace umožňuje měnit limit procenta podobnosti u jednotlivých analýz. **Výstupní podmínky**: Aplikace zobrazila výsledek analýz.

Případ užití: Uložení výsledků
Vstupní podmínky: Analýza byla úspěšně dokončena.
Tok událostí: 1. PU začíná, poklepnutím na tlačítko **Export výsledků** v PU Zobrazení výsledků. 2. Aplikace vyžaduje vybrání externího textového souboru pro uložení výsledku analýz. 3. Uživatel vybere soubor pro uložení výsledku analýz. 4. Aplikace uloží výsledky analýz do vybraného souboru. **Výstupní podmínky**: Aplikace uložila výsledky analýz.

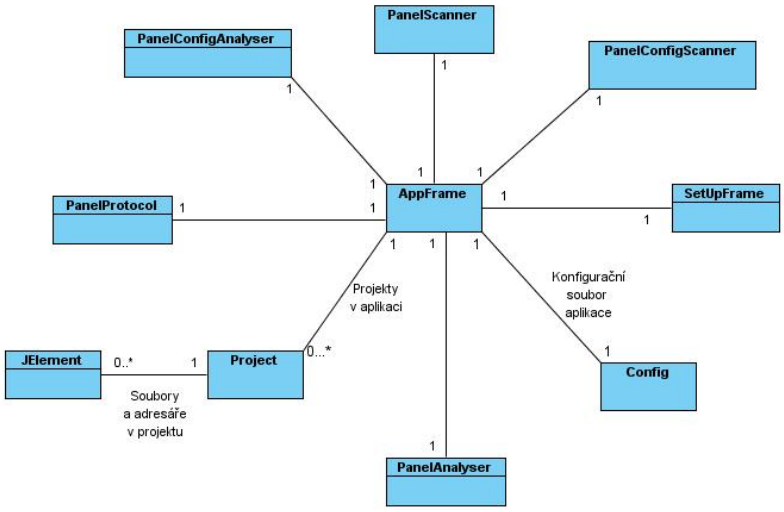
Případ užití: Nastavení aplikace
Vstupní podmínky: Analýza ještě nebyla spuštěna.
Tok událostí: 1. PU začíná, poklepnutím na tlačítko **Nastavení** v hlavním okně aplikace. 2. Aplikace umožňuje nastavení široké škály parametrů pro jednotlivé analýzy. 3. Uživatel nastaví požadované parametry a potvrdí změny. 4. Aplikace si uloží nastavené parametry do vnitřní paměti. **Výstupní podmínky**: Parametry analýz byly nastaveny.
Alternativní tok: 1. Uživatel nastaví požadované parametry a zvolí možnost **Uložit jako výchozí**. 2. Aplikace uloží nastavené parametry do externího souboru, aby je mohla načíst při každém spuštění. **Výstupní podmínky**: Parametry byly uloženy do externího souboru.

2.3 Analytický model

Na analytickém modelu (Obrázek 3) se snažím zachytit podstatné třídy a vztahy mezi nimy.

Základní třídou je třída *AppFrame* reprezentující hlavní okno aplikace. Tato třída obsahuje konfigurančí soubor (třída *Config*) a informace o jednotlivých projektech obsažených ve třídách *Project)*.

Každý projekt obsahuje jeden nebo více *Element*ů, ve kterých jsou uloženy informace o jednotlivých souborech a adresářích.



Obrázek 3: Analytický model

PanelScanner provádí načtení pracovního adresáře a uložení informací o projektech do externích XML souborů. Třída *PanelAnalyser* zajišťuje načtení projektů z XML souborů a provedení analýz.

Třída *PanelProtokol* pak zajišťuje zobrazení výpisu podezřelých projektů a jeho export do textového souboru.

Pro nastavení aplikace slouží třída *PanelSetUp*.

3 Použité technologie, externí knihovny a druhy analýz

3.1 Technologie

3.1.1 Java SE

Java SE je zkratka pro *Java Platform Standard Edition*. Je to základní platforma pro vývoj a nasazení programů v jazyku Java od firmy *Sun Microsystems*. Díky *Java Virtual Machine* (JVM) - virtuální stroj Javy je Java programovacím jazykem pro tvorbu aplikací použitelných na různých platformách. Toto je jeden z důvodů, proč jsem se rozhodl vytvořit svou aplikaci právě v programovacím jazyku Java.

Základ Javy SE tvoří technologie *Application Programming Interfaces* (APIs), což je rozhraní pro programování aplikací. Existují dva pilíře této platformy [3]:

- *Java SE Runtime Environment* (JRE) - prostředí obsahující komponenty, knihovny, JVM, čímž umožňuje spuštění aplikací napsaných v javě.
- *Java SE Development Kit* (JDK) - Sada vývojových nástrojů, který obsahuje samotné JRE a nástroje, jako například javac - kompilátor jazyka Java.

Nevýhodou použití technologií Java je samozřejmě nutností uživatele mít nainstalovanou JDK. Pro vývoj aplikace jsem použil Java SE verze 5.1.7.

3.1.2 Swing a jeho integrace v IDE

Jednou ze základních součástí Javy je Swing [4], což je API pro tvorbu grafického uživatelského prostředí *Graphical User Interface*(GUI). Jeho předchůdcem je *Abstract Window Toolkit*(AWT), které je součástí JDK od verze 1.0 (Swing od verze 1.2). AWT poskytuje méně rozhraní a tříd, čímž je sice jednodužší, ale zároveň poskytuje méně možností a funkcionality. Jelikož Swing vychází právě z AWT, je na něm do jisté míry závislý. Důkazem je používání knihovny jako *java.awt.event*. Swing je také postaven na základech některých AWT tříd (např. *Component*) [2].

Tvorba GUI pomocí knihoven Swing je jednodužší při použití nástroje Swing GUI Builder v *NetBeans IDE* [5]. NetBeans je *open-source* (OOS) prostředí pro vývoj softwaru, a to nejen pro jazyk Java. Swing GUI Builder umožňuje jednotlivé komponenty GUI aplikací libovolně rozmístit, spravovat a pracovat s nimi pomocí myši v návrhovém módu, namísto pracného psaní kódu, který nástroj automaticky generuje za nás. Kód pak samozřejmě můžeme doplnit manuálně dle potřeby. Na základě mých dobrých zkušeností s NetBeans prostředím a jeho dobrou integrací knihovny Swing jsem se rozhodl použít tento nástroj pro tvorbu mé aplikace.

Možnou alternativou k AWT a Swing je *StandardWidget Toolkit* (SWT) od *Eclipse Foundation*.

3.1.3 UML

Unified Modeling Language (UML) je standartní modelovací jazyk pro specifikaci, konstrukci a dokumentaci artefaktů systémů s převážně softwarovou charakteristikou [6]. Pomocí UML můžeme zachytit různé fáze vývoje byznis modelování od analýzy, přes modelování až k nasazení a údržbě systému. Nástroje tohoto jazyka jsou použity jako nosný pilíř pro analýzu a návrh nástroje mého programu. Diagramy prezentovány v mojí práci byly vytvořeny v programu *Visual Paradigm for UML* ve verzi 6.4 [7].

3.1.4 XML

Zkratka XML znamená *EXtensible Markup Language*, což by se doslovně dalo přeložit jako rozšiřitelný značkovací jazyk. XML je obecně značkovací jazyk [8], který umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat. Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí.

Jelikož je formát XML ideální pro uchování, přenášení a výměnu dat, rozhodl sem se jej použít pro uložení konfigurace programu JProjectComparator a pro ukládání informací z načtených projektů pro rychlé zpracování a snadnou manipulaci.

3.2 Externí knihovny

3.2.1 Knihovny pro práci s dokumenty

Jelikož moje aplikace musí umět extrahovat text z textových dokumentů, importoval sem do aplikace knihovny podporující práci s různými typy dokumentů.

Pro dokumenty Microsoft Word je to knihovna *poi-3.5-beta4* [9].

Pro PDF dokumenty knihovna *PDFBox-0.7.3* [10].

A pro práci s dokumenty formátu ODT jsem použil knihovnu *jOpenDocument-1.1b3* [11].

Použil jsem také knihovnu fontů *FontBox-0.1.0* [12].

3.2.2 Knihovna String Similarity

Pro porovnávání obsahu textů jsem vybral algoritmus String Similarity, jenž je součástí importované knihovny *simpack-0.91* [13]. V této variantě algoritmu je propočítávána míra vzdáleností mezi dvěmy řetězci.

3.3 Použité druhy analýz

Analýzu projektů - tedy samotné porovnávání projektu s ostatnímy projekty a určení jeho podobnosti vůči nim, jsem se rozhodl rozdělit do tří základních částí.

Jelikož jsou součástí odevzdávaných prací zejména textové dokumenty, orientoval jsem dvě části na práci s dokumenty a jednu část na porovnávání obecných souborů a adresářů. Mezi formáty textových dokumentů, se kterými aplikace bude umět pracovat, jsem zvolil dokumenty Portable Document Format (.pdf) a dokumenty, tabulky a prezentace Microsoft Word (přípony .doc a .docx) a Open Document (.odt).

3.3.1 Souborová analýza

První část je věnována základním parametrům souborů, které si o nich systém uchovává. Vybral jsem z nich název, velikost souboru a datum jeho poslední modifikace. Datum vytvoření souboru je pozměněno už jen při překopírování projektu do jiné složky a pro porovnávání tedy není vhodné. Navíc u operačního systému Linux tato informace není uchovávána vůbec.

3.3.2 Analýza přehledu dokumentu

Ve druhé části analýzy jsem se rozhodl zkoumat základní parametry dokumentů jako jsou autor, titul, apd. Při pozdějších testech jsem zjistil, že informace dokumentů ukládány operačním systémem Windows (DocumentSummaryInformation) jsou omezeny pouze na tento operační systém při použití systému souborů NTFS a při přenosu mezi počítači (FLASH disk, email) nejsou uchovávány. Byl sem tedy nucen tuto část aplikace přepracovat a soustředit se na informace uchovávány jednotlivými editory. Jelikož jsou tyto informace u různých formátů různé, není možno porovnávat mezi sebou rozdílné formáty dokumentů.

3.3.3 Analýza obsahu dokumentu

Poslední část je věnována porovnávání textu obsaženém v samotným dokumentech. Pro toto porovnání jsem si zvolil dvě základní metody - Indexovou analýzu a algoritmus String Similarity.

3.3.3.1 Indexová analýza Metoda porovnávání textu indexovou analýzou spočívá ve vytvoření tabulky slovníku pro každý text. Ta bude obsahovat všechny obsažené slova v textu a četnost jeho výskytu. Jednotlivé tabulky se pak porovnávají mezi sebou. Tuto metodu lze též aplikovat i na četnost písmen a znaků.

3.3.3.2 algoritmus String Similarity Algoritmus String Similarity je algoritmus pro vyhodnocení podobnosti textu podle pseudokódu Williama E. Winklera. Algoritmus je založen na hledání podobných řetězců a hledání vzdálenosti mezi nimi.

Tento algoritmus je velice vhodný pro porovnávání dvou prostých textů. Nelze jej však aplikovat pro porovnávání zdrojových kódů aplikací. Různé programy totožného programovacího jazyka obsahují spousty shodných řetězců (instrukce, metody apd.), které se navíc vyskytují v malých vzdálenostech od sebe. Pro porovnávání zdrojových kódů by bylo potřeba implementovat do aplikace algoritmus orientovaný pro tuto problematiku.

4 Návrh

V Návrhové části se přechází od analytických modelů, které jsou méně podrobné, příliš všeobecné a neuvádějící detaily, k o mnoho podrobnějším modelům. Úlohou návrhových modelů je podat natolik podrobný pohled na problémovou oblast, aby byla možná její následná implementace.

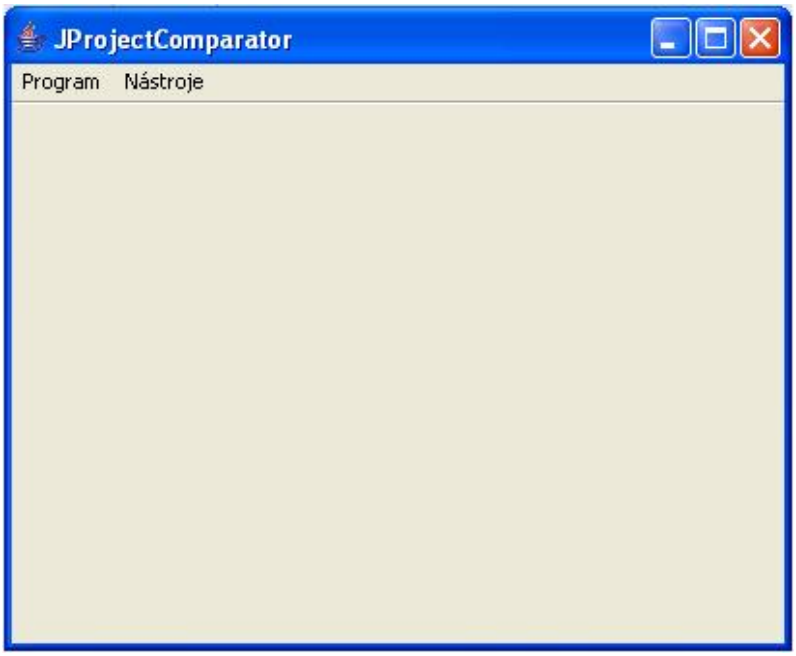
4.1 Návrh GUI

Grafické rozhraní aplikace jsem se rozhodl ralizovat pomocí hlavního okna aplikace obsahující menu, do kterého budou vkládány panely realizující jednotlivé případy užití. Jedná se o panel pro nastavení scanneru, nastavení analýzy, provedení scanneru, provedení analýzy, zobrazení a export výsledků. Pro nastavení aplikace bude sloužit vedlejší okno.

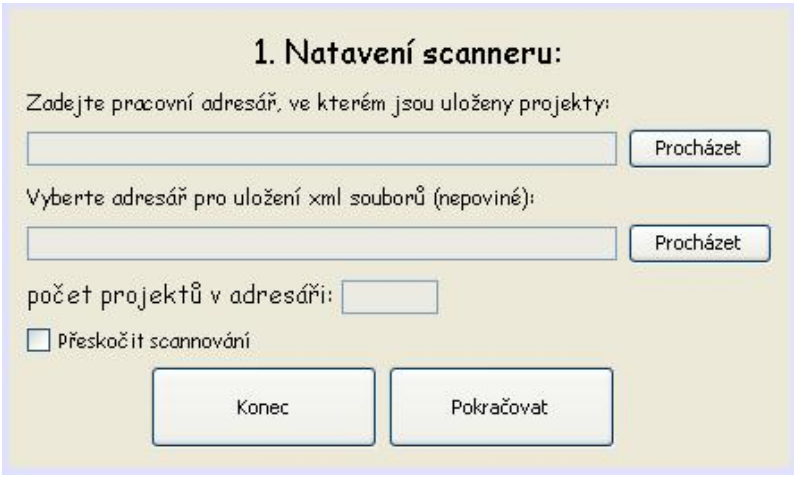
Následující obrázky tedy znázorňují prototyp vzhledu aplikace. Vznikl ve fázi, kdy ještě nebyly naimplementovány žádné třídy. Návrh jsem spracoval v prostředí *NetBeans*. Jednotlivé tlačítka a funkce v této části návrhu ještě nebyli funkční, šlo jen o grafický obal aplikace.

Návrh GUI je zobrazen na Obrázcích 4 až 10.

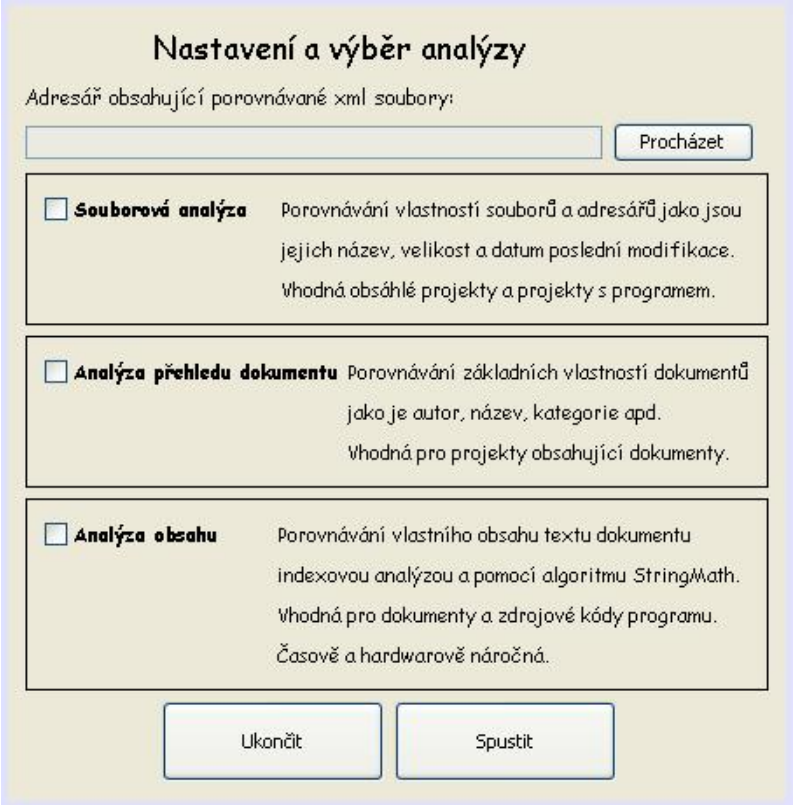
Pro konečné GUI aplikace bylo krom elementárních úprav nutno přehodnotit okno pro nastavení aplikace, které jsem pro objemnost rozdělil do panelu se záložkami.



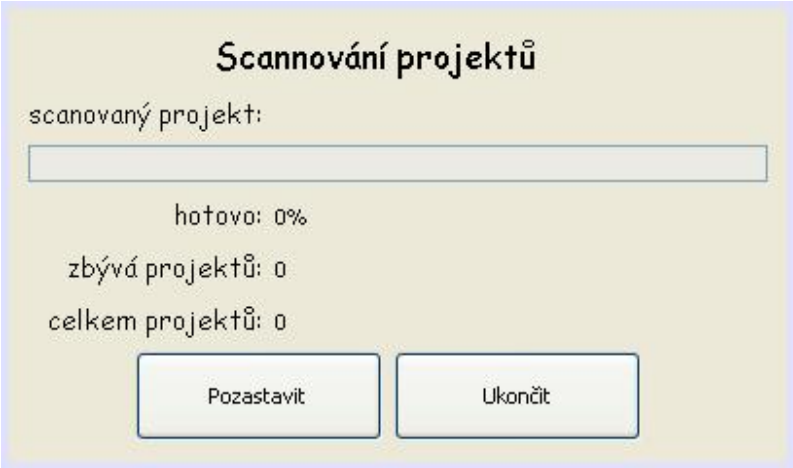
Obrázek 4: Hlavní okno aplikace JProjectComparator



Obrázek 5: Panel Nastavení scanneru

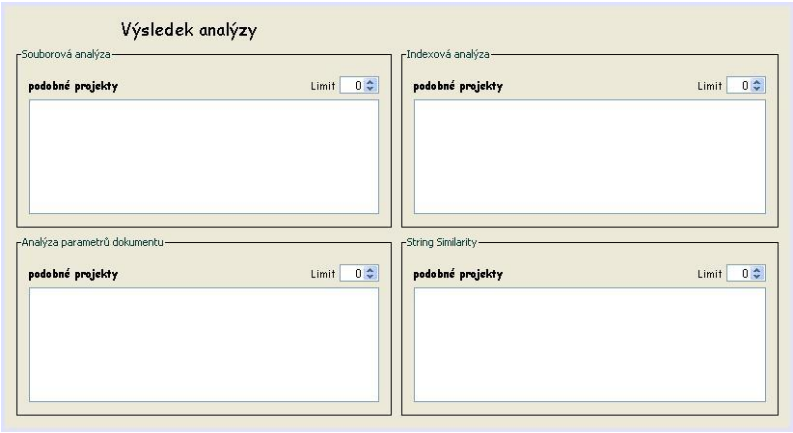


Obrázek 6: Panel Nastavení analýzy

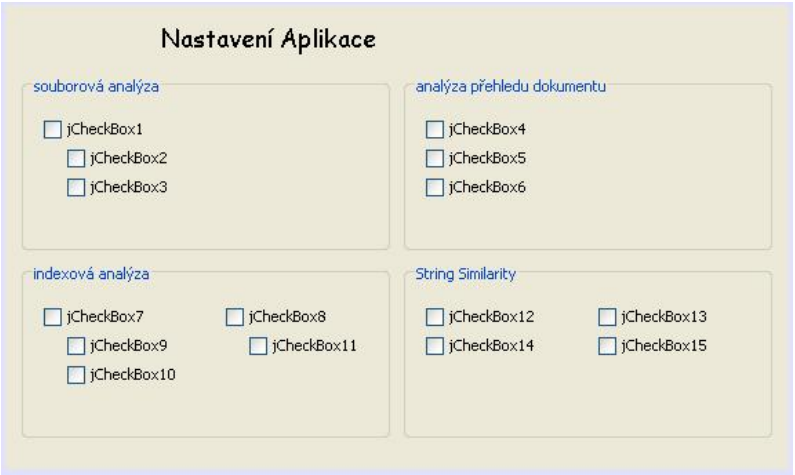


Obrázek 7: Panel Spuštění scanneru

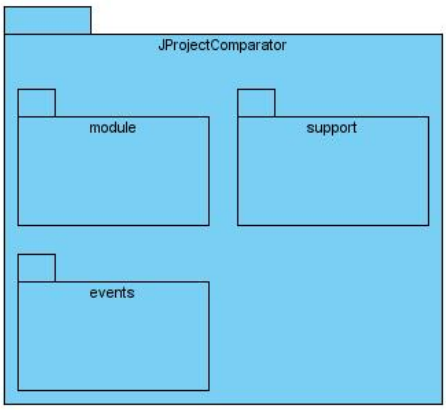
Obrázek 8: Panel Spuštění analýzy



Obrázek 9: Panel Zobrazení výsledků



Obrázek 10: Okno Nastavení Aplikace



Obrázek 11: Rozdělení tříd do balíčků

4.2 Diagram tříd - Návrhový model

Jádro návrhu Nástroje JProjectComparator se opírá o diagram tříd, na kterým je zachycena byznys logika aplikace. Při jeho tvorbě jsem vycházel z již provedené analýzy. Jelikož ve fázi návrhu se proniká do větších podrobností, jsou i v diagramech určité změny a spřesnění.

Třídou reprezentující projekty je třída *Project*. Projekty načítá a porovnává třída PanelAnalyser z externích XML souborů vygenerovanými pomocí třídy PanelScanner. Tyto třídy rovněž zajišťují vykreslení grafických prvků na panelu. Při těchto činnostech je třeba zohlednit, že mohou zabírat netriviální časový úsek a zároveň by aplikace měla poskytovat informaci o jeho průběhu. Z uvedených důvodů jsem se rozhodl oddělit samotné provedení scannaru a analyzéru z hlavního vlákna ve kterém se vykresluje GUI do vedlejšího vlákna. Logickou návaznost provedení operací a zobrazení panelů aplikace zajišťuje třída *AppFrame*.

Jelikož okno pro nastavení *SetUpFrame* bylo nutno rozdělit do tří záložek, bylo třeba vytvořit další tři třídy *PanelFileAnalyse*, *PanelSummaryInformationAnalyse* a *PanelContentAnalyse*, které obsahují pouze GUI.

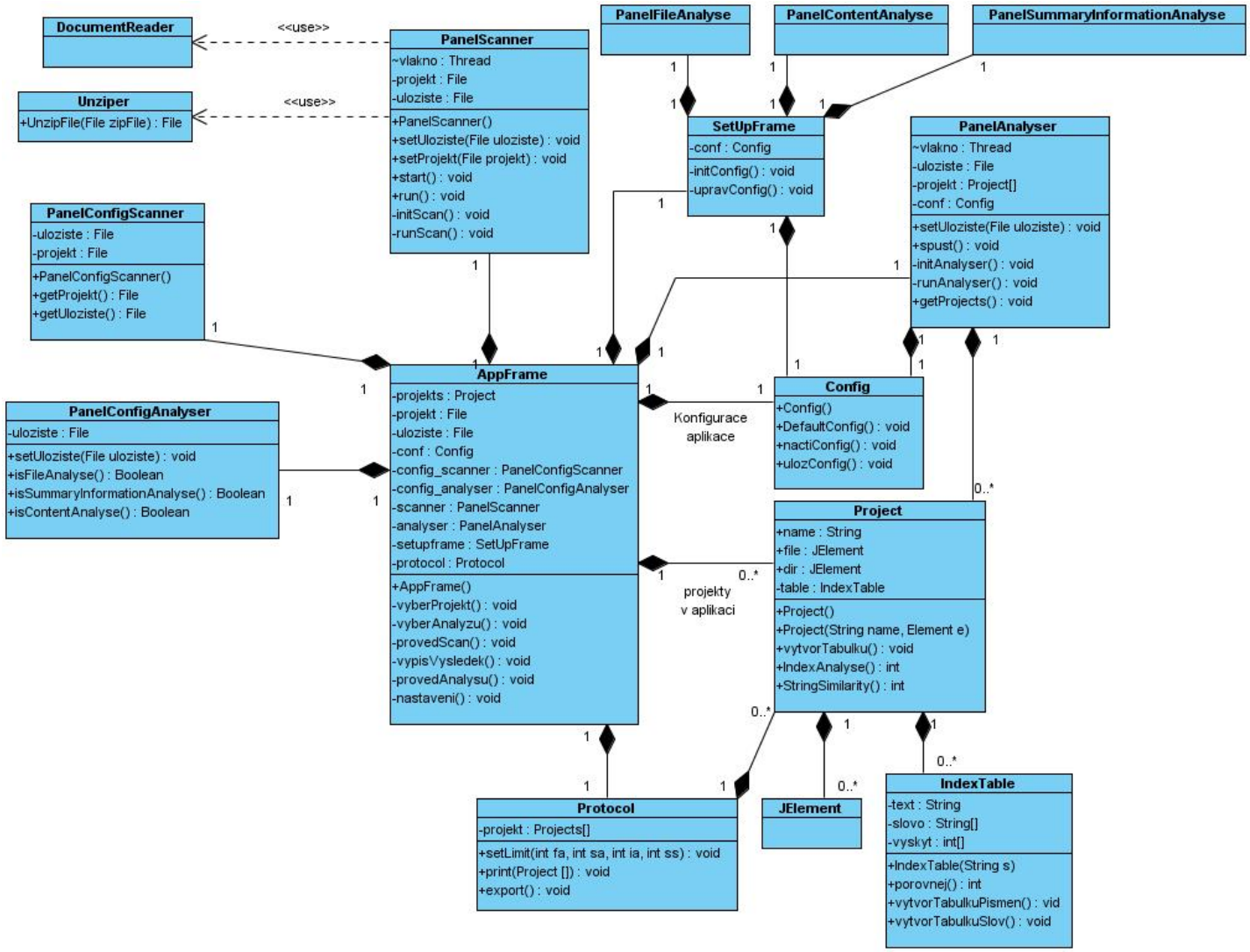
Třída *DocumentReader* slouží pro extrakci textu a informací z dokumentů. Pro četnost metod v ní obsažených jsem se rozhodl je na diagram návrhového modelu nazachytit. Stejné rozhodnutí jsem učinil vůči atributům třídy *JEmelent*. Tyto atributy uchovávají informace o dokumentech nebo souboru či adresáři obecně.

Třída *Unziper* je určena pro extrakci archivů formátu zip.

Návrhový model je zachycen na Obrázku 12

4.3 Rozdělení tříd do balíčků

Pro lepší přehled při programování jsem rozdělil třídy do balíčků. Hiearchii balíčků lze vidět na Obrázku 11. Balíček *module* obsahuje všechny výkoné a grafické části aplikace. Balíček *support* je určen pro podpůrné třídy aplikace a v balíčku *events* jsou třídy pro vytvořené události.



Obrázek 12: Návrhový model

4.4 Struktura XML souborů

4.4.1 Konfigurační soubor

Jelikož výsledek samotné analýzy je z velké části závislý na nastavení parametrů uživatelem, rozhodl jsem se ukládat zvolené nastavení do externího XML souboru uloženého v kořenovém adresáři aplikace. Po každém spuštění aplikace se soubor automaticky načte. Struktura XML souboru je rozdělené do tří základních elementů, podle tří základních analýz, jež se v aplikaci provádí. Jedná se o souborovou analýzu (*File_analyse*), analýzu přehledu dokumentu (*Summary_analyse*) a analýzu obsahu dokumentu (*Content_analyse*). V rámci těchto elementů jsou pak ukládány informace potřebné k provedení dané analýzy.

XML soubor lze modifikovat buďto přímo zápisem do něj, nebo pomocí nastavení v aplikaci. V případě že XML soubor není v kořenovém adresáři uložen nebo je poškozen, vygeneruje aplikace nastavení s původnímy hodnotami.

Struktura externího XML souboru je znázorněna na Výpisu 1.

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
    <File_analyse>
        <Name up = "true" >
            <suffix  up = "true"  />
            <dir up = "false"  />
            <preference size = "30" />
        </Name>
        <Length up = "true" >
            <dir up = "false"  />
            <preference size = "60" />
            <tolerance size = "0"  />
        </Length>
        <Date up = "true" >
            <dir up = "false"  />
            <preference size = "70" />
        </Date>
        <Limit size  = "20"/>
    </File_analyse>
    <Summary_analyse>
        <doc>
            <Author up = "true"/>
            <Category up = "true"/>
            <Comments up = "true"/>
            <Company up = "true"/>
            <Keywords up = "true"/>
            <Manager up = "true"/>
            <Subject up = "true"/>
            <Title  up = "true"/>
        </doc>
        <docx>
            <Category up = "true"/>
            <Creator up = "true"/>
            <Description up = "true"/>
            <Keywords up = "true"/>
            <Language up = "true"/>
```

```
            <Subject up = ”true”/>
            <Title  up = ”true”/>
            <Version up = ”true”/>
        </docx>
        <pdf>
            <Author up =”true”/>
            <Keywords up =”true”/>
            <Producer up =”true”/>
            <Subject up =”true”/>
            <Title  up =”true”/>
        </pdf>
        <odt>
            <Creator up =”true”/>
            <Description up =”true”/>
            <Language up =”true”/>
            <Subject up =”true”/>
            <Title  up =”true”/>
        </odt>
        <Limit size  =”20”/>
    </Summary_analyse>
    <Content_analyse>
        <Index_analyse>
            <letters  up =”true”>
                <additional up =”true”/>
                <numbers up =”true”/>
            </letters>
            <words up =”true”>
                <numbers up =”true”/>
                < filter  up=”false”/>
            </words>
            <limit  size  =”90”/>
        </Index_analyse>
        < String_similarity  up =”true”>
            <additional up =”false”/>
            <numbers up =”false”/>
            <space up =”false”/>
            <limit  size  =”50”/>
        </ String_similarity >
    </Content_analyse>
</Configuration>
```

Výpis 1: struktura konfiguračního souboru

4.4.2 Soubory s informacemi o projektech

Informace o souborech a dokumentech obsažených v projektu ukládá scanner do XML souborů. Základním elementem takto vytvořeného XML souboru je element *Basedir*. Tento element představuje adresář, ve kterém je projekt uložen. Dále pak tento element obsahuje další elemnty reprezentující adresáře a soubory uložené v něm.

Hiearchické uspořádání elementů odpovídá hiearchickému uspořádání souborů. Elementy pak nesou informace o těchto souborech a adresářích. Je-li soubor zároveň dokument podporovaného typu, obsahuje navíc informace o dokumentu.

Příklad takového XML souboru je na Výpisu 2.

```
<?xml version="1.0" encoding="utf-8"?>
<Basedir name="ID5" patch="C:\Documents and Settings\Ateus\Plocha\Pokus\ID5">
  <file  name="Dokument.pdf" length="29184" date="1237397697343">
    <type>doc</type>
    <type>pdf</type>
    <Author><![CDATA[Ladislav Stehno]]></Author>
    <Keywords><![CDATA[test]]></Keywords>
    <Producer><![CDATA[OpenOffice.org 3.0]]></Producer>
    <Subject><![CDATA[]]></Subject>
    <Title><![CDATA[testovací pdf]]></Title>
    <text><![CDATA[Nějaký text]]></text>
  </file>
</Basedir>
```

Výpis 2: struktura souboru obsahující informace o projektech

5 Implementace a testování

V této části kapitoly popisuji implementační kapitoly nejdůležitějších tříd a metod v programu JProjectComparator. Textový popis je vhodně doplněn úryvky ze zdrojového kódu. Na konci kapitoly popisuji testování aplikace.

5.1 Implementace hlavního okna

Hlavní okno programu je zastoupeno třídou *JFrame*. Tato třída má také za úkol zajistit logické pořadí provedení úkolů. Panely jsou do okna vkládány pomocí metody *add(Component c)*. Pro ilustraci jsem uvedl vložení panelu pro nastavení scanneru a pro spuštění scanneru.

Ukončení funkce panelu aplikace zachycuje dvěma způsoby. První možnost je stiskem tlačítka uživatelem. Druhou možností je ukončení vlákna. Pro tento účel jsem implementoval událost zachycující ukončení vlákna ve Výpisu 3.

```
public class AppFrame extends JFrame implements ActionListener, ThreadEndListener{
    private PanelConfigScanner config_scanner;
    private PanelScanner scanner;

    //vložení panelu pro nastavení scanneru
    private void nastaveniScanneru(){
        add(config_scanner);
        pack();
        config_scanner.bt_spustit.addActionListener(this);
    }

    //vložení panelu pro scanner
    private void spustScanner(){
        scanner.setUloziste(uloziste);
        scanner.setProjekt(projekt);
        scanner.addThreadEndListener(this);
        add(scanner);
        pack();
        scanner.start();
    }

    //zachycení události generované při stisku tlačítka
    public void actionPerformed(ActionEvent e) {
            if(e.getSource().equals(config_scanner.bt_spustit)){    //ukončení panelu pro nastavení
                scanner
            ...
            }
    }
    public void ThreadEnd(ThreadEndEvent e) {
            if(e.getVlakno().equals(scanner.vlakno)){//ukončení vlákna scanneru
            ...
            }
    }
}
```

Výpis 3: vložení panelu do hlavního okna

5.2 Načtení informací o souborech a dokumentech

Informace o souborech a adresářích jsou získány pomocí metod třídy java.io.File. Zisk Informací o dokumentech zprostředkovává třída DocumentReader. Tato třída pracuje se 4 základními formáty souborů (doc, docx, ppt, pptx, xls, odt, odp, ods, pdf). Pro každý formát obsahuje metodu pro získání textu z dokumentu a metody pro zisk dodatečných informací o dokumentech. Na Výpisu 4 jsou znázorněny pro ukázku metody pro extrakci textu z těchto 4 typů dokumentů.

```
public class DocumentReader {
   public String prevodHWPT(File file) throws IOException{//extrakce textu z˜dokumentů doc
        FileInputStream fis = new FileInputStream(file.getPath());
        HWPFDocument doc = new HWPFDocument(fis);
        WordExtractor we = new WordExtractor(doc);
        return we.getText();
  }
  public String prevodXSSF(File file) throws Exception{//extrakce textu z˜dokumetů docx
        Package pck=Package.open(file.toString());
        XWPFDocument doc = new XWPFDocument(pck);
        XWPFWordExtractor xpf=new XWPFWordExtractor(doc);
        pck.close();
        return xpf.getText();
  }
  public String prevodPDF(File file) throws IOException{//extrakce texttu z˜dokumentu pdf
        PDDocument doc = null;
        String text = null;
        try{
            doc = PDDocument.load( file );
            PDFTextStripper textStripper = new PDFTextStripper();
            text = textStripper.getText(doc);
        }
        finally{
            if( doc != null )
                doc.close();
        }
        return text;
  }
  public String prevodODT(File file) throws IOException {//extrakce textu z˜dokumentů odt
        String text = "";
        ODPackage pkg = new ODPackage(file);
        ODXMLDocument xml = pkg.getContent();
        Element e = (Element)((Element) xml.getDocument().getRootElement().getChildren().get(3))
             .getChildren().get(0);
        for(int i=0; i<e.getChildren().size(); i++)
           text +=(((Element)(e.getChildren().get(i))).getValue())+"\n";
        return text;
  }
}
```

Výpis 4: extrakce textu z dokumentu

5.3 Čtení XML souboru

Pro načtení XML souborů využívám třídy *org.w3c.dom.Document*. Ten pak zavoláním *getDocumentElement()* vrací hlavní element dokumentu. Získávání informací z tohoto zdrojového elementu pak docílíme za pomocí metod *getChildNodes()* pro získání seznamu potomků daného elemntu, *getTextContent()* pro získání textu v elementu a *getAttributes()* pro získání atributů daného elementu. Jako ukázku uvádím ve Výpisu 5 úsek kódu třídy *Config*, která ve své metodě *nactiConfig* využívá pro načtení informací z konfiguračního souboru právě práci s XML soubory.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
Document doc = null;
DocumentBuilder builder = dbf.newDocumentBuilder();
doc = builder.parse(new File("config.xml"));
Element e = doc.getDocumentElement();

fa_name = Boolean.parseBoolean(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(0).getAttributes().item(0).getNodeValue());
fa_name = Boolean.parseBoolean(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(0).getChildNodes().item(0).getAttributes().item(0).getNodeValue
    ());
fa_name_dir = Boolean.parseBoolean(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(0).getChildNodes().item(1).getAttributes().item(0).getNodeValue
    ());
fa_name_preference = Integer.valueOf(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(0).getChildNodes().item(2).getAttributes().item(0).getNodeValue
    ());
fa_length = Boolean.parseBoolean(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(1).getAttributes().item(0).getNodeValue());
fa_length_dir = Boolean.parseBoolean(e.getElementsByTagName("File\_analyse").item(0).
    getChildNodes().item(1).getChildNodes().item(0).getAttributes().item(0).getNodeValue
    ());
```

Výpis 5: práce s XML souborem

Metoda *getElementsByTagName("File_analyse")* vrací seznam všech elementů s názvem *File_analyse*. Jelikož v konfiguračním souboru je pouze jeden takový element, zajímá nás hned první položka v seznamu *item(0)*. Metoda *getNodeValue* vrací hodnotu atributu.

5.4 Oddělení činnosti z hlavního vlákna

Java umožňuje tzv. *multithreading* neboli paralelní běh dvou či více částí programu. Každá paralelně běžící část programu se v Javě nazývá vlákno (*thread*). Na většině dnešních počítačů s jedním procesorem se samozřejmě nejedná o fyzicky současný běh vláken, ale jednotlivá vlákna se na procesoru střídají.

Jelikož samotná funkce scanneru a analýzy mohou trvat velmi dlouho dobu, je potřeba pro jejich činnost vytvořit nové vlákno, aby neovlivňovali vykreslování GUI.

Do těchto dvou tříd jsem tedy naimplementoval rozhraní *Runnable*, která definuje jádro vlákna, metodu *Run()*. Příklad jsem demonstroval na Výpisu 6

```
public class PanelScanner extends javax.swing.JPanel implements Runnable{
    Thread vlakno;

    public void start () {
            vlakno = new Thread(this);
        vlakno. start () ;
    }

    public void run(){
        initScan () ;
        runScan();
        ThreadEndEvent event = new ThreadEndEvent(this, vlakno);
        fireThreadEnd(event);//událost generovaná při ukončení vlákan
    }
}
```

Výpis 6: Oddělení činnosti z hlavního vlákna

Spuštění vlákna provedeme zavoláním metody *start()* na instanci třídy PanelScanner.

5.5 Testování

Testování nástroje JProjectComparator jsem uskutečňoval inkrementálním způsobem v různých fázích jeho implementace. Jakmile jsem vytvořil nějakou funkcionalitu, snažil jsem se ji dostatečně otestovat. Pro potřeby ověření programu jsem si vytvořil sadu několika projektů s rúznými soubory a textovými dokumenty. Do testování jsem zapojil i jiné osoby, abych odhalil více náhodných a nepředvídatelných stavů. Aplikaci jsem spouštěl na více platformách (Windows XP Professional Edition,WindowsVista Home Premium, Ubuntu 8.04 Desktop Edition).

6 Uživatelská příručka

6.1 Instalace a spuštění aplikace JProjectComparator

- Nainstalujte do operačního systému JDK.
- Nakopírujte z CD adresář JProjectComparator do libovolného umístění na lokálním disku.
- Spusťte dávkový soubor **start.exe**, který je uložen v adresáři JProjectComparator.

6.2 Nastavení scanneru

Po spuštění programu můžete zadat cestu k pracovnímu adresáři s projekty a vybrat adresář pro uložení vygenerovaných XML souborů. Aplikaci lze také ukončit tlačítkem **Konec**, nebo vybráním volby **Konec** v menu **Prgram**.

- Chcete-li zadat cestu k adresáři obsahující projekty, otevřte dialogové okno stiskem tlačítka **Procházet** pod výzvou pro zadání této cesty. V políčku **počet projektů v adresáři** se zobrazí počet projektů ve vybraném adresáři.
- Pokud chcete proces skenování přeskočit, zaškrtněte políčko **Přeskočit scannování**.
- Je-li potřeba, zadejte cestu adresáře, do kterého se mají ukládat vygenerované XML soubory. Cestu zadáte poklepnutím na tlačítko **Procházet** pod výzvou k zadání cesty k adresáři pro uložení externích XML souborů.
- Pro pokračování k nastavení analýzy stiskněte tlačítko **Pokračovat**. V případě že jste zaškrtli možnost **Přeskočit scannování**, pro pokračování stiskněte **Přeskočit**.

6.3 Nastavení analýzy

- Je-li to potřeba, zadejte cestu k adresáři obsahující XML soubory k porovnávání. Cestu zadáte vyvoláním dialogového okna stisknutím tlačítka **Procházet**.
- Zaškrtněte analýzy, které chcete provést. Pro spuštění scanneru a analýzy stiskněte tlačítko **Spustit**.

6.4 Scannování

Scannování probíhá automaticky. Lze jej kdykoliv pozastavit stiskem tlačítka **Pauza** a poté pokračovat v práci tlačítkem **Pokračovat**. Aplikace poskytuje informace o právě skenovaném projektu (**cesta**), procentu provedené práce (**hotovo**), zbývajících projektech (**zbývá projektů**) a o celkovém počtu projektů (**celkem projektů**).

6.5 Analýza

Analýza probíhá automaticky ihned po dokončení scanneru. Lze jí kdykoliv pozastavit stiskem tlačítka **Pauza** a poté pokračovat v práci tlačítkem **Pokračovat**. Aplikace poskytuje informace o právě skenovaném projektu (**aktuální projekt**), procentu provedené práce (**hotovo celkem**), zbývajících projektech (**zbývá projektů**), o celkovém počtu projektů (**celkem projektů**) a o počtu nalezených podezřelých projektů (**podezřelé projekty**).

6.6 Zobrazení výsledků

Panel pro zobrazení výsledků se zobrazí ihned po dokončení analýzy. Podezřelé projekty se obrazí v textovém panelu dle dané analýzy. V menu **Nástroje** se zpřístupní volba **export**, která vyvolá dialogové okno pro vybrání textového souboru, do kterého se výsledky uloží. Ukázka exportních výsledků je na Výpisu 7.

```
Výsledek Souborové analýzy:
ID2.xml je  podobný s ID7.xml na 100%

Výsledek Analýzy přehledu dokumentu:
ID1.xml je  podobný s ID2.xml na 20%
ID2.xml je  podobný s ID3.xml na 100%
ID3.xml je  podobný s ID8.xml na 100%

Výsledek Indexové analýzy
ID2.xml je  podobný s ID7.xml na 100%

Výsledek String  Similarity
ID2.xml je  podobný s ID7.xml na 100%
ID3.xml je  podobný s ID8.xml na 59%
```

Výpis 7: Ukázka exportího výsledku

6.7 Nastavení

Nastavení aplikace vyvoláte pomocí volby **Nastavení** v menu **Nástroje**. Mezi nastavením jednotlivých analýz se pak přepínáme poklepáváním na záložky dané analýza.

7 Závěr

Výsledkem práce je funkční program, který ulehčí práci jeho uživatelům. Díky výběru vhodných vývojových prostředí a volby programovacího jazyku Java jsem docílil jednoduché a platformově přenositelné aplikace. Podrobnou analýzou jsem odhalil různé specifické případy, které mohou při použití aplikace nastat. Rozvážný návrh umožňuje možnou rozšiřitelnost nástroje o nové způsoby funkcionality. Uživatelské rozhraní umožňuje intuitivní a uživatelsky přátelské prostředí. Práce poskytuje popis jednotlivých vývojových etap programu. Tento popis by mohl sloužit jako návod pro přístup k tvorbě nového softwaru.

Pokračováním práce by mohlo být rozšíření funkcionality progamu. Vhodným doplněním by mohlo být třeba rozšíření pro práci s dalšími formáty textových dokumentů a pro práci s jinými kompresními formáty. Analýza by mohla být také rozšířena o algoritmy pro porovnávání zdrojových kódů. V úvahu přichází i možná lokalizace aplikace do jiných jazyků.

8 Reference

[1] J., Bloch. *Java efektivně. 57 rad softwarového experta*. Praha Grada Publishing, 2006 232 s.

[2] PAVEL, Herout. *Java - grafické uživatelské prostředí a čeština*. České Budějovice : Kopp, 2006. 320 s.

[3] Sun Microsystems, Inc. *Java SE at a Glance* [online]. 1994 [cit. 2009-01-22]. Dostupný z WWW:

http://java.sun.com/javase/technologies/.

[4] Sun Microsystems, Inc. *Swing (JavaTM Foundation Classes)* [online]. 2005 [cit. 2009-01-22]. Dostupný z WWW:

http://java.sun.com/javase/6/docs/technotes/guides/swing/index.html.

[5] *NetBeans IDE* [online]. [cit. 2009-01-22]. Dostupný z WWW:

http://www.netbeans.org/.

[6] Object Management Group, Inc. *UML® Resource Page* [online]. 1997 [cit. 2009-01-22]. Dostupný z WWW:

http://www.uml.org/.

[7] Visual Paradigm. *Increase productivity, communication, and collaboration using UML visual modeling platform* [online]. 2004 [cit. 2009-01-22]. Dostupný z WWW:

http://www.visual-paradigm.com/.

[8] W3C®. *Extensible Markup Language (XML)* [online]. 1996 [cit. 2009-01-22]. Dostupný z WWW:

http://www.w3.org/XML/.

[9] The Apache Software Foundation. *Apache POI - Java API To Access Microsoft Format Files* [online]. 2002 [cit. 2009-01-22]. Dostupný z WWW:

http://poi.apache.org/.

[10] The Apache Software Foundation. Apache PDFBox - Java PDF Library [online]. 2008 [cit. 2009-01-22]. Dostupný z WWW:

http://incubator.apache.org/pdfbox/.

[11] JOpenDocument.org. *JOpenDocument Homepage. Open Document Library* [online]. 2009 [cit. 2009-01-22]. Dostupný z WWW:

http://www.jopendocument.org/.

[12] SourceForge, Inc. *SourceForge.net: FontBox* [online]. 1999 [cit. 2009-01-22]. Dostupný z WWW:

http://sourceforge.net/projects/fontbox/.

[13] Dynamic & Distributed information Systems Group * University of Zurich. *SimPack Project Page* [online]. 1999 [cit. 2009-01-22]. Dostupný z WWW:

http://www.ifi.uzh.ch/ddis/simpack.html.